568727 v1
M-7949 US

EXPRES[illegible]AIL LABEL NO:
EL 375 516 754 US

ALTERNATE SET OF REGISTERS TO SERVICE CRITICAL INTERRUPTS AND OPERATING SYSTEM TRAPS

Philip A. Bourekas

FIELD OF THE INVENTION

The present invention relates to a microprocessor based electronic system and in particular to servicing exceptions including hardware interrupts and operating system calls.

BACKGROUND

Conventional digital computers execute tasks by executing sequences of related instructions that access or modify the contents of a set of registers and memory cells. Occasionally, during execution of a sequence of instructions an exception occurs, such as an interrupt or operating system calls (traps), which requires the operating system to immediately respond. A processor may receive exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, and I/O interrupts. Conventionally, when an exception is detected, the normal sequence of instructions is suspended while the processor deals with the exception. The use of exceptions and exception processing is well known in the art of computer architecture.

Fig. 1 is a flow chart 10 of processing an exception in a conventional manner. As shown in Fig. 1, when an exception is asserted (block 12) the exception is decoded and execution diverted (using an exception "vector") to a new location that is designated for handling exceptions (block 14). The processor and/or system software must determine the priority of the exception (block 16) particularly if there are multiple exceptions, in which case the highest priority must be dealt with first. The execution of the original task is suspended and the key processor "state" information must be preserved in main memory (block 18) so that once the exception is serviced, the original task may be resumed. The exception is then serviced (block 20). After the exception is serviced, the processor must restore the original "state" information that was earlier preserved (block 22). The execution of the original task is then resumed (block 24).

Typically, in an embedded real-time system, a number of exceptions, with varying levels of importance, i.e., priority, must be managed. As can be seen in Fig. 1, decoding the exception, determining priority, and preserving the state of the system, present a large amount of overhead in servicing exceptions.

One conventional strategy implemented in processors to speed operation, typically when handling multiple exceptions, is called "nesting". To reduce the amount of time spent in prioritization of multiple interrupts, a processor may use a fixed set of priorities. When a higher priority interrupt is signaled (higher than the current executing program), the processor suspends execution and automatically preserves the entire processor state into an area of memory referred to as the "stack". Unfortunately, as processors have developed, the amount of state information has grown such that saving all processor state, regardless of need, is an undue burden. Consequently, stacking in current processors actually degrades performance due to the amount of state information that must be stacked.

Another technique employed to service exceptions includes register windows, which help minimize the time spent doing "context switching." In a register window approach, windows are made available by periodically allocating a set of registers from a larger physical set when a subroutine is called or an exception is signaled. Unfortunately, depending on the number of physical registers and the program, register "spills" may occur during which there are not enough registers to handle the exception. Consequently, prior windows have to be written into main memory to create a new window resulting in a degradation in the performance.

Thus, what is needed is a technique to minimize the overhead associated with servicing an exception to speed the handling of exceptions and to separate especially critical exceptions from normal priority exceptions.

SUMMARY

In accordance with an embodiment of the present invention, a processor includes a set of general purpose registers and a set of "alternate", but otherwise general purpose, exception registers, e.g., eight registers, that are switched for a subset of general purpose registers and are used while servicing specific exceptions. The set of exception registers is dedicated for servicing high priority, i.e., critical, exceptions. The processor may provide a dedicated vector, which, when used, also turns on the set of exception registers for an asserted exception. Software conventions, such as an API, can allocate different portions of the set of exception registers or different sets of exception registers for servicing different types of exceptions, such as interrupts operating system calls (traps), while separate dedicated vectors may be used for the interrupts and the operating systems calls.

In one embodiment, select logic circuits may be used to enable and disable the general purpose registers and the exceptions registers. Thus, for example, a select logic circuit coupled to the exception registers may receive an exception register active bit from the opcode of the instruction, while a select logic circuit coupled to the general purpose registers may receive an inverted exception register active bit from the opcode. Thus, one of either the general purpose registers and the exception registers will be enabled and one disabled. The select logic circuits may also receive the same register address bits from the opcode of the instruction.

In accordance with another embodiment of the present invention, a method of interrupting the execution of a task for servicing an exception in a processor, includes asserting an exception, diverting execution to a vector address and activating logic to use the exception alternate registers, rather than general purpose registers, servicing the exception using the set of exception registers and swapping out the exception registers for the general purpose registers before resuming execution of the original task. Because servicing the exception now utilizes the exception registers without disrupting the state of the interrupted task, there is no need for explicit state management prior to servicing the exception. The processor may automatically activate exception registers and use dedicated vectors with the exception registers. The dedicated vector may be used for high priority, i.e.,

critical, exceptions, while another vector, for exceptions which use the general purpose registers, is used for exceptions with a lower priority. In addition, the method may include providing separate dedicated vectors for high priority interrupts and high priority operating system calls, while another vector is used with the general purpose registers for lower priority exceptions.

Because the exception registers are automatically activated for fast exceptions, there is no need to determine the priority of the exception. With the use of a dedicated vector or dedicated vectors for interrupts and operating system calls, there is no need to decode the exception. Advantageously, during the servicing of the exception, the values of the exception registers may be modified, without disrupting the state of the interrupted task. Thus, because a set of dedicated exception registers are swapped in for the general purpose registers to service an exception, there is no need for explicit state management prior to and subsequent to servicing the exception.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a flow chart of a conventional process used to service to an exception.

Fig. 2 is a block diagram of register files showing the programmers view of the processor register file under the invention while servicing the high priority, i.e., critical, exceptions and/or normal exceptions in accordance with an embodiment of the present invention.

Fig. 3 shows a block diagram of a circuit, in accordance with an embodiment of the present invention, using a set of general purpose registers and an alternate set of dedicated exception registers.

Fig. 4 is flow chart showing the process of handling an exception in accordance with the an embodiment of the present invention.

Fig. 5 is another flow chart showing the process of servicing an exception in accordance with an embodiment of the present invention

DETAILED DESCRIPTION

A processor, in accordance with an embodiment of the present invention, speeds the service of high priority, i.e., critical, exceptions, such as interrupts and operating system calls (traps), by providing an alternate set of registers for the service routine to use while servicing the exception. By providing alternate registers to service the exception, the entire CPU instruction set is available to service the exception and there is no overhead for state preservation or recovery at the beginning and end of servicing the exception. Moreover, if the exception is one in a series, such as receiving cells in a packet, intermediate results, e.g., for a current subset of the packet, do not need to be pushed to memory and other information, while other incoming parameters and outgoing results, can be made available to the exception service routine and to the rest of the software environment, without operating system traps or exceptions.

A processor, in accordance with an embodiment of the present invention, divides exceptions into two levels: "normal" for lower priority exceptions, and "fast" for high priority or critical exceptions. Normal exceptions are dealt with in a conventional way, such as using a conventional software approach. Software decodes the current exception, applies its own prioritization rules and relies on software to save as little or much of the state information as required for the current exception.

A fast exception has a higher priority than a normal exception. In other words, if both are asserted at the same time, the fast exception is recognized and serviced first. The servicing of normal exceptions is not pre-empted by fast exceptions. Fast exceptions are serviced with a set of exception registers that automatically replace at least a subset of general purpose registers and a dedicated vector is used to begin exception service. Thus, fast exceptions are automatically separated from normal exceptions.

Fig. 2 is a block diagram showing the programmers view of the processor register files 100 showing the servicing of fast exceptions and normal exception in accordance with an embodiment of the present invention. While executing tasks and during operation on "normal" exceptions, general purpose registers GP(31:16)

102, GP(15:8) 104, and GP(7:0) 106 are used, as indicated by arrows 103 and 105. In accordance with an embodiment of the present invention, however, a set of dedicated fast exception registers ER(15:8) 108 are included as an alternate to the set of general purpose registers GP(15:8) 104. Thus, during a fast exception, general purpose registers GP(15:8) 104 are swapped out for exception register ER(15:8) 108, as indicated by arrows 107, 109.

With the use of a set of fast exception registers ER(15:8) 108, there remains a number, e.g., 24, of other registers available so that the exception handler, operating system, and user tasks can share data without being required to go to memory to obtain the rest of the system state. In an embodiment of the present invention, interrupts and traps may be allocated to different portions of the set of exception registers, e.g., half the exception registers may be allocated to the fast interrupts while the other half of the exception registers are allocated to the fast traps. Alternatively, different sets of exception registers may be used for fast interrupts and fast traps.

It should be understood that the specific number of registers shown in Fig. 2, including the number of exception registers ER(15:8) 108 is illustrative, and any number of registers may be used. Further, if desired, multiple alternate sets of exception registers may be used. Thus, multiple exceptions may be mapped to multiple exception registers, e.g., exception (1) is mapped to a first set of alternate exception registers, exception (2) is mapped to a second set of alternate exception registers, etc.... Moreover, separate interrupt and trap vectors may be associated with each alternate set of exception registers. Cache locking may also be used to hold exception vectors. Cache locking is a method of marking certain cache lines such that they are not selected for replacement during cache refill.

Because a set of alternate registers are swapped in during high priority exceptions, there is no need for explicit state management. In other words, the state of general purpose registers GP(15:8) 104 is maintained in those registers while the exception is serviced in the exception registers ER(15:8) 108. The exception handler can freely modify the values in the exception registers without disrupting the state of the interrupted task that is present in the general purpose registers 104. Further, because the same set of alternate general purpose registers are used each time a fast exception is signaled, there is no possibility of a register window spill, and results can be preserved between exceptions. Thus, the operating system can use the values in the exception registers as intermediate results, thereby minimizing memory reads and writes at the beginning and end of exceptions, which are relatively costly in terms of performance.

The alternate exception registers ER(15:8) 108 may be substituted out, and general purpose registers GP(15:8) 104 back in, by a single instruction at the end of the exception handling. Consequently, the code that is required to explicitly restore the operating state of the register file at the end of the exception and to resume execution of the original task is eliminated.

In addition, because a dedicated vector is used for fast exceptions, which have a higher priority than normal exceptions, fast exceptions are automatically separated from normal exceptions. Consequently, there is no need to decode the fast exception or to determine its priority.

Fig. 3 shows a block diagram of a circuit 150, in accordance with an embodiment of the present invention, using exception registers 152, labeled as upper upper register bank 152 and general purpose registers 154, 156, and 158, labeled as lower upper register bank 154, upper lower register bank 156, and lower lower register bank 158, respectively.

Each instruction can specify up to three registers; and a processor can be multiple issue (multiple instructions per cycle). Commonly, the processor register file supports numerous simultaneous read and/or write accesses. For example, a dual issue machine would need to support four read accesses and two write accesses per clock cycle into the register file, while control logic keeps the multiple writes, or a write/read pair, from interfering with each other. Each read or write "port" has logic to select the physical register that it is using. Thus, each port has "select" logic, which utilizes a register address field to pick the register out of the register "file".

Thus, as shown in Fig. 3, registers banks 152, 154, 156, and 158 are each selected, via a port SEL that is coupled to a select logic, shown in Fig. 3 as AND logic gates. Logic gate 160 controls the selection of upper upper register bank 152, i.e., the exception register, while logic gates 162 and 164 control the selection of lower upper register bank 154, logic gate 166 controls the selection of upper lower register bank 156, and logic gate 168 controls the selection of lower lower register bank 158. Low order bits of RegAddr pick the specific register within the bank. Each logic gate 160, 162, 164, 166, and 168 receives on input terminals (some of which are inverted, as shown in Fig. 3) the third register bit (RegAddr (3)) and the fourth register bit (RegAddr(4)). As indicated in Fig. 3, logic gate 162 enables registers 31 through 24 of upper lower register bank 154, while logic gate 164 enables registers 23 through 16 of upper lower register bank 154.

The register bits RegAddr(3) and RedAddr(4) come from the opcode for the instruction being executed. Thus, for example, the architecture of circuit 150 supports a "3 operand" format, which is well understood by those of ordinary skill in the art, whereby an opcode can specify two source registers and a destination (e.g. reg(a) <- reg(b) op reg(c), where a, b, and c are register numbers (not necessarily different...)). Thus, the opcodes contain the specification of the register numbers, or register address. If desired, some processor architectures or implementations may implement performance techniques to cause the specification of the register numbers, or register address, that finally reach a register file to come from some buffering or control store, rather than the current instruction entering the pipeline.

In accordance with the present invention, circuit 150 must also factor in whether the alternate exception registers, i.e., upper upper registers bank 152, are currently active. Thus, logic gates 160 and 166, which control the upper upper register bank 152 and upper lower register bank 156, respectively, also receive an exception register active bit (ExcRegAct). As shown in Fig. 3, logic gate 166, which controls the general purpose register upper lower bank 156, receives an inverted exception register active bit (ExcRegAct), and thus is disabled if the exception register active bit is high. Logic gate 160, which controls exception register upper upper bank 152, however, receives the exception register active bit (ExcRegAct), and thus is enabled when the exception register active bit is high.

Logic gates 162, 164, and 168 do not receive the exception register active bit (ExcRegAct) because the lower upper register bank 154 and lower lower register bank 158 do not care if the exception register is active. The exception register active bit (ExcRegAct) is set by a fast exception and cleared when the exception is returned from.

Table 1 below summarizes the logic of circuit 150, where an "X" indicates "do not care".

ExcRegAct	RegAddr (4)	RegAddr (3)	Physical Register	Select Activated
X	0	0	GP[7:0] 158	Lower Lower
0	0	1	GP[15:8] 156	Upper Lower
X	1	0	GP[23:16]154	Lower Upper
X	1	1	GP[31:24]154	Lower Upper
1	0	1	ER[15:8]152	Upper Upper

Table 1

It should be understood that the function of circuit 150 may be implemented in other ways. For example, an out of order architecture may implement a technique called "register renaming", whereby the current location of a given physical register may be dynamically moved in a larger, "virtually addressed" register file. In that case, the logic shown in Fig. 3 would be used to properly determine the virtual name of interest, and then provide a select signal as an input to the mapping logic for the register access, where an exception register active bit would choose an alternative virtual name from that chosen if the exception register is not active.

Using the alternate set of exception registers, in accordance with the present invention, eliminates many of the conventional steps that must be performed by the system when handling an exception. Fig. 4 is flow chart showing the process 200 of handling an exception in accordance with the present invention. When an exception occurs, the control status register is set to an appropriate value depending on the type of exception (202). In addition, a check is performed to determine if there is an exception within an exception. The processor is forced into Kernel Mode and interrupts are disabled. The base exception vector address bit (BEV) is then checked (204), which determines the base address as shown in blocks 206 and 208. The process then checks to see if the fast interrupt is enabled (210) to determine the offset for the address. If the fast interrupt is enabled and signaled the offset is chosen, as shown in block 212, which swaps the exception registers ER(15:8) for the general purpose registers GP(15:8), as discussed in reference to Figs. 2 and 3. If the fast interrupt is not enabled or not asserted, the exception may be handled in a conventional manner, shown as Cause.IV in block 214, which decides between the offsets shown in blocks 216 and 218. The program counter (PC), which is the address of the current instruction, is then the base, as determined at blocks 204 plus the offset as determined at blocks 210 and 214. The exception is then diverted to the appropriate vector.

Fig. 5 is another flow chart showing the process 300 of servicing an exception in accordance with an embodiment of the present invention. Once the exception is asserted (302), the process determines whether the exception is designated as "fast" (304). As discussed above, this may be accomplished by recognizing that the feature is enabled (a bit is set in a control register) and looking at the exception , and thus does not require explicit software decoding of the exception. If the exception is "fast", a set of general purpose registers are swapped with a set of exception registers (306). The exception is then serviced (308), which is followed by swapping out the exception registers (310) and resuming execution of the original task.

If, on the other hand, the exception is not designated as "fast" as determined in step 304, the exception may be handled in a conventional manner. For example, as shown in Fig. 5, the exception is decoded (312), and priority is determined (314). The processor state information is preserved prior (316) prior to servicing the exception (318). Once the exception is serviced, the original state information is restored (320) and the execution of the original task is resumed (322). Of course, any desired method of handling the exception, including software techniques, may be used if the exception is not designated as "fast".

While the present invention has been described in connection with specific embodiments, one of ordinary skill in the art will recognize that various substitutions, modifications and combinations of the embodiments may be made after having reviewed the present disclosure without departing from the scope of the invention. The specific embodiments described above are illustrative only. Thus, the spirit and scope of the appended claims should not be limited to the foregoing description.